

AUSTRALIAN OS9 NEWSLETTER

sides 'No. of cylinders' (in decimal) :Interleave value: (in decimal) @FREE Syntax: Free [devname] Usage : Displays number of free sectors on a device @GFX Syntax: RUN GFX(<func><args>) Usage : Graphics interface package for BASIC09 to do compatible VDG graphics commands @GFX2 Syntax: RUN GFX2([path]<func><args>) Usage : Graphics interface package for BASIC09 to handle

Usage : window help to @IDENT from OS single line directory @INKE input a the proc memory

EDITOR

Gordon Bentzen (07) 344-3681

SUB-EDITOR

Bob Devries (07) 372-7816

TREASURER

Don Berrie (07) 375-1284

LIBRARIAN

Jean-Pierre Jacquet (07) 372-4675

SUPPORT

Fax Messages (07) 372-8325

Brisbane OS9 Users Group

text files @LOAD Syntax: Load <pathname> [e] Usage : Loads modules into memory @MAKDIR Syntax: Makdir <pathname> Usage : Creates a new directory file @MDIR Syntax: Mdir [e] Usage : Displays the present memory module directory Opts : e = print extended module directory @MERGE Syntax: Merge <path>

@MFREE Syntax

@MODPATCH

memory from c

compare modul

to module C o

module M = ma

Usage : Set m

monochrome m

and links an OS

Procs [e] Usage

display all proc

current data directory path

execution directory path

module> Usage : BASIC09 run time package @SETIME Syntax: Setime

[yy/mm

Syntax:

num @

@TMODE Syntax: Tmode [pathname] [params] Usage : Displays or changes

the operating parameters of the terminal @TUNEPOR

[value] Adjust the baud value for the serial port @UNLINK Syntax: Unlink

<modname> Usage : Unlinks module(s) from memory @WCREATE Syntax:

Addresses for Correspondence

Editorial Material:

Gordon Bentzen

8 Odin Street

SUNNYBANK Qld 4109

Subscriptions & Library Requests:

Jean-Pierre Jacquet

27 Hampton Street

DURACK Qld 4077

Volume 6

June 1992

Number 5

@TMODE Syntax: Tmode [pathname] [params] Usage : Displays or changes the operating parameters of the terminal @TUNEPOR Tuneport </t1 or /p> [value] Adjust the baud value for the serial port @UNLINK Syntax: Unlink <modname> Usage : Unlinks module(s) from memory @WCREATE Syntax:

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 6 Number 5

EDITOR : Gordon Bentzen
SUBEDITOR : Bob Devries

TREASURER : Don Berrie
LIBRARIAN : Jean-Pierre Jacquet

SUPPORT : Brisbane OS9 Level 2 Users Group.

Those of you who are also supporters of the Australian CoCo-Link magazine would no doubt be aware that CoCo-Link will cease production with the December 1992 edition. We are of course saddened to hear this news and would like to offer our congratulation to Robbie Dalzell, Garry Holder, Sub-Editors and the other helpers for the great job they have done over the past few years. Their genuine efforts to help CoCo users and to keep the CoCo community alive is to be commended, well done.

We have covered many times our disappointment with the Tandy decision to quit the CoCo and I don't want to go into that again. It is however quite an achievement that CoCo Usergroups and magazines have been able to continue for a couple of years after Tandy dropped the product AND their support.

The Australian OS9 Usergroup continues on with the promise of bigger and better things in the future. OS9 level 2 on the CoCo3 & 6809 is still hard to beat in many ways. The humble CoCo is certainly not able to match the speed, video displays and many other beautiful things on offer in the 68xxx OS9 machines but the CoCo3 with Microware's OS9 L2 has set the others a real challenge in such things as the L2 windowing system.

We believe that the new OS9 68K platforms such as the MM/1, Delmar and Frank Hogg's machines will ensure a growing following of OS9 and maybe even OS9000 on those 386 and 486 P.C's.

OS9 Usergroups in one form or another are on the increase. Information about the OCN Community on Fido has been featured in previous editorials, as well as this we

are seeing talk of the old U.S. OS9 Usergroup being reformed with many of those notable names involved.

Of course it may be that we will also have to fold up one day, but that will be a decision that you, the members, will make. Until then we plan to bring news and articles of interest to help all members enjoy OS9.

IN THIS EDITION

Our tutorial in the 'C' programming language continues with Chapter 8. The Rainbow OS9 article index continues with the 1990 listing.

Bob Devries has generously supplied yet another 'C' programme, complete with source code. This programme will convert an Amiga Sound Sample file so that it may be used with the OS9 'Play'.

A disk reader/verifier programmer in 'C', also by Bob Devries is included.

We have all seen the 'More' utility in use under MS-DOG or the 'C' version on a CoCo. Well in this edition we have the source for an assembler version by Boisy Pitre for you to type in.

We do hope that you find something useful in this edition and remind you that our P.D. library continues to grow slowly with all sorts of interesting and useful utilities and programmes from overseas and local authors. All these and our UGCAT (Catalogue of Usergroup P.D. software) are available for our usual nominal copy fee.

Cheers, Gordon.

oooooooooooo0000000000oooooooooooo

A C Tutorial Chapter 8 - Pointers

WHAT IS A POINTER?

Simply stated, a pointer is an address. Instead of being a variable, it is a pointer to a variable stored somewhere in the address space of the program. It is always best to use an example so load the file named POINTER.C and display it on your monitor for an example of a program with some pointers in it. For the moment, ignore the data declaration statement where we define "index" and two other fields beginning with a star. It

is properly called an asterisk, but for reasons we will see later, let's agree to call it a star. If you observe the first statement, it should be clear that we assign the value of 39 to the variable "index". This is no surprise, we have been doing it for several programs now. The next statement however, says to assign to "pt1" a strange looking value, namely the variable "index" with an ampersand in front of it. In this example, pt1 and pt2 are pointers, and the variable "index" is a simple variable.

Now we have a problem. We need to learn how to use pointers in a program, but to do so requires that first we define the means of using the pointers in the program. The following two rules will be somewhat confusing to you at first but we need to state the definitions before we can use them. Take your time, and the whole thing will clear up very quickly.

TWO VERY IMPORTANT RULES

The following two rules are very important when using pointers and must be thoroughly understood.

1. A variable name with an ampersand in front of it defines the address of the variable and therefore points to the variable. You can therefore read line six as "pt1 is assigned the value of the address of index".
2. A pointer with a "star" in front of it refers to the value of the variable pointed to by the pointer. Line nine of the program can be read as "The stored (starred) value to which the pointer "pt1" points is assigned the value 13". Now you can see why it is convenient to think of the asterisk as a star, it sort of sounds like the word store.

MEMORY AIDS

1. Think of & as an address.
2. Think of * as a star referring to stored.

Assume for the moment that "pt1" and "pt2" are pointers (we will see how to define them shortly). As pointers, they do not contain a variable value but an address of a variable and can be used to point to a variable. Line six of the program assigns the pointer "pt1" to point to the variable we have already defined as "index" because we have assigned the address of "index" to "pt1". Since we have a pointer to "index", we can manipulate the value of "index" by using either the variable name itself, or the pointer. Line nine modifies the value by using the pointer. Since the pointer "pt1" points to the variable "index", then putting a star in front of the pointer name refers to the memory location to which it is pointing. Line nine therefore assigns to "index" the value of 13. Anyplace in the program where it is permissible to use the variable name "index", it is also permissible to use the name "&pt1" since they are identical in meaning until the pointer is reassigned to some other variable.

ANOTHER POINTER

Just to add a little intrigue to the system, we have another pointer defined in this program, "pt2". Since "pt2" has not been assigned a value prior to statement

seven, it doesn't point to anything, it contains garbage. Of course, that is also true of any variable until a value is assigned to it. Statement seven assigns "pt2" the same address as "pt1", so that now "pt2" also points to the variable "index". So to continue the definition from the last paragraph, anyplace in the program where it is permissible to use the variable "index", it is also permissible to use the name "&pt2" because they are identical in meaning. This fact is illustrated in the first "printf" statement since this statement uses the three means of identifying the same variable to print out the same variable three times.

THERE IS ONLY ONE VARIABLE

Note carefully that, even though it appears that there are three variables, there is really only one variable. The two pointers point to the single variable. This is illustrated in the next statement which assigns the value of 13 to the variable "index", because that is where the pointer "pt1" is pointing. The next "printf" statement causes the new value of 13 to be printed out three times. Keep in mind that there is really only one variable to be changed, not three. This is admittedly a very difficult concept, but since it is used extensively in all but the most trivial C programs, it is well worth your time to stay with this material until you understand it thoroughly.

HOW DO YOU DECLARE A POINTER?

Now to keep a promise and tell you how to declare a pointer. Refer to the third line of the program and you will see our old familiar way of defining the variable "index", followed by two more definitions. The second definition can be read as "the storage location to which "pt1" points will be an int type variable". Therefore, "pt1" is a pointer to an int type variable. Likewise, "pt2" is another pointer to an int type variable. A pointer must be defined to point to some type of variable. Following a proper definition, it cannot be used to point to any other type of variable or it will result in a "type incompatibility" error. In the same manner that a "float" type of variable cannot be added to an "int" type variable, a pointer to a "float" variable cannot be used to point to an integer variable. Compile and run this program and observe that there is only one variable and the single statement in line 9 changes the one variable which is displayed three times.

THE SECOND PROGRAM WITH POINTERS

In these few pages so far on pointers, we have covered a lot of territory, but it is important territory. We still have a lot of material to cover so stay in tune as we continue this important aspect of C. Load the next file named POINTER2.C and display it on

your monitor so we can continue our study. In this program we have defined several variables and two pointers. The first pointer named 'there' is a pointer to a 'char' type variable and the second named 'pt' points to an 'int' type variable. Notice also that we have defined two array variables named 'strg' and 'list'. We will use them to show the correspondence between pointers and array names.

A STRING VARIABLE IS ACTUALLY A POINTER

In the programming language C, a string variable is defined to be simply a pointer to the beginning of a string. This will take some explaining. Refer to the example program on your monitor. You will notice that first we assign a string constant to the string variable named 'strg' so we will have some data to work with. Next, we assign the value of the first element to the variable 'one', a simple 'char' variable. Next, since the string name is a pointer by definition of the C language, we can assign the same value to 'two' by using the star and the string name. The result of the two assignments are such that 'one' now has the same value as 'two', and both contain the character 'T', the first character in the string.

Note that it would be incorrect to write the ninth line as 'two = *strg[0];' because the star takes the place of the square brackets. For all practical purposes, 'strg' is a pointer. It does, however, have one restriction that a true pointer does not have. It cannot be changed like a variable, but must always contain the initial value and therefore always points to its string. It could be thought of as a pointer constant, and in some applications you may desire a pointer that cannot be corrupted in any way. Even though it cannot be changed, it can be used to refer to other values than the one it is defined to point to, as we will see in the next section of the program. Moving ahead to line 12, the variable 'one' is assigned the value of the ninth variable (since the indexing starts at zero) and 'two' is assigned the same value because we are allowed to index a pointer to get to values farther ahead in the string. Both variables now contain the character 'a'.

The C programming language takes care of indexing for us automatically by adjusting the indexing for the type of variable the pointer is pointing to. In this case, the index of 8 is simply added to the pointer value before looking up the desired result because a 'char' type variable is one byte long. If we were using a pointer to an 'int' type variable, the index would be doubled and added to the pointer before looking up the value because an 'int' type variable uses two bytes per value stored. When we get to the chapter on structures, we will see that a variable can have many, even into the hundreds or thousands, of characters per variable, but

the indexing will be handled automatically for us by the system. Since 'there' is already a pointer, it can be assigned the value of the eleventh element of 'strg' by the statement in line 16 of the program. Remember that since 'there' is a true pointer, it can be assigned any value as long as that value represents a 'char' type of address. It should be clear that the pointers must be 'typed' in order to allow the pointer arithmetic described in the last paragraph to be done properly. The third and fourth outputs will be the same, namely the letter 'c'.

POINTER ARITHMETIC

Not all forms of arithmetic are permissible on a pointer. Only those things that make sense, considering that a pointer is an address somewhere in the computer. It would make sense to add a constant to an address, thereby moving it ahead in memory that number of places. Likewise, subtraction is permissible, moving it back some number of locations. Adding two pointers together would not make sense because absolute memory addresses are not additive. Pointer multiplication is also not allowed, as that would be a funny number. If you think about what you are actually doing, it will make sense to you what is allowed, and what is not.

NOW FOR AN INTEGER POINTER

The array named 'list' is assigned a series of values from 100 to 199 in order to have some data to work with. Next we assign the pointer 'pt' the value of the 28th element of the list and print out the same value both ways to illustrate that the system truly will adjust the index for the 'int' type variable. You should spend some time in this program until you feel you fairly well understand these lessons on pointers. Compile and run POINTER2.C and study the output. You may recall that back in the lesson on functions we mentioned that there were two ways to get variable data back from a function. One way is through use of the array, and you should be right on the verge of guessing the other way. If your guess is through use of a pointer, you are correct. Load and display the program named TWOWAY.C for an example of this.

FUNCTION DATA RETURN WITH A POINTER

In TWOWAY.C, there are two variables defined in the main program 'pecans' and 'apples'. Notice that neither of these is defined as a pointer. We assign values to both of these and print them out, then call the function 'fixup' taking with us both of these values. The variable 'pecans' is simply sent to the function, but the address of the variable 'apples' is sent to the function. Now we have a problem. The two arguments are not the same, the second is a pointer to a variable. We must

somehow alert the function to the fact that it is supposed to receive an integer variable and a pointer to an integer variable. This turns out to be very simple. Notice that the parameter definitions in the function define "nuts" as an integer, and "fruit" as a pointer to an integer. The call in the main program therefore is now in agreement with the function heading and the program interface will work just fine. In the body of the function, we print the two values sent to the function, then modify them and print the new values out. This should be perfectly clear to you by now.

The surprise occurs when we return to the main program and print out the two values again. We will find that the value of pecans will be restored to its value before the function call because the C language makes a copy of the item in question and takes the copy to the called function, leaving the original intact. In the case of the variable "apples", we made a copy of a pointer to the variable and took the copy of the pointer to the function. Since we had a pointer to the original variable, even though the pointer was a copy, we had access to the original variable and could change it in the function. When we returned to the main program, we found a changed value in "apples" when we printed it out. By using a pointer in a function call, we can have access to the data in the function and change it in such a way that when we return to the calling program, we have a changed value of data. It must be pointed out however, that if you modify the value of the pointer itself in the function, you will have a restored pointer when you return because the pointer you use in the function is a copy of the original. In this example, there was no pointer in the main program because we simply sent the

address to the function, but in many programs you will use pointers in function calls. One of the places you will find need for pointers in function calls will be when you request data input using standard input/output routines. These will be covered in the next two chapters. Compile and run TWOWAY.C and observe the output.

POINTERS ARE VALUABLE

Even though you are probably somewhat intimidated at this point by the use of pointers, you will find that after you gain experience, you will use them profusely in many ways. You will also use pointers in every program you write other than the most trivial because they are so useful. You should probably go over this material carefully several times until you feel comfortable with it because it is very important in the area of input/output which is next on the agenda.

PROGRAMMING EXERCISES

1. Define a character array and use "strcpy" to copy a string into it. Print the string out by using a loop with a pointer to print out one character at a time. Initialize the pointer to the first element and use the double plus sign to increment the pointer. Use a separate integer variable to count the characters to print.
2. Modify the program to print out the string backwards by pointing to the end and using a decrementing pointer.

oooooooooooooooooooooooooooooooo

CoCo-Link

CoCo-Link is an excellent magazine to help you with the RSDOS side of the Colour Computer. It is a bi-monthly magazine published by Mr. Robbie Dalzell. Send your subscriptions to:

CoCo-Link
31 Nedlands Crescent
Pt. Noarlunga Sth.
South Australia
Phone: (08) 3861647

oooooooooooooooooooooooooooooooo

An Index of Rainbow OS9 Articles
compiled by Bob Devries
January - December '90

January 1990 page 110
One Label or Two - A simple label-maker program.
Kevin Deneen

Breakpoint - Don't spend all day looking for a certain disk.
Greg Law

January 1990 page 56

January 1990 page 117

AUSTRALIAN OS9 NEWSLETTER

KISSable OS9 - Talking about modularity.
Dale L. Puckett

February 1990 page 56
Drive With Speed - Verify makes your OS9 floppy drives
work much faster.
Stephen B. Goldberg

February 1990 page 122
Breakpoint - Examining a technique for parsing command
line parameters in C.
Greg Law

March 1990 page 86
Do the Split - Making large files easier to handle.
Stephen B. Goldberg

March 1990 page 96
Breakpoint - Giving it top priority.
Greg Law

March 1990 page 65
KISSable OS9 - Tales of a new improved OS9 Level II for
the Color Computer 3.
Dale L. Puckett

April 1990 page 44
Predicting Profits - A helpful program for those planning
a career in business.
David Macias

April 1990 page 113
Breakpoint - Talking about conversions and diversions.
Greg Law

April 1990 page 52
KISSable OS9 - A new look for OS9.
Dale L. Puckett

May 1990 page 46
Larry's Labeler - Organizing your floppy disks.
Larry Pittman

May 1990 page 30
Breakpoint - Directories and OS9 file structure.
Greg Law

June 1990 page 36
Type Does Windows - A utility to switch window types.
Chris Swinefurth

June 1990 page 44
Breakpoint - Scaling the directory tree.
Greg Law

July 1990 page 32
For Your Eyes Only - Logon protection for the single-user

OS9 system.
Stephen B. Goldberg

July 1990 page 72
Breakpoint - The missing link.
Greg Law

July 1990 page 48
KISSable OS9 - Loan procedures for MVFinance.
Dale L. Puckett

August 1990 page 20
CoCo 3 GIME CART* IRQs Explained - A software technique
that eliminates the cartridge interrupt problem.
Bruce Isted.

August 1990 page 64
Living Without Line Numbers - Will the virtues of Basic09
never cease?
Dean Bergmann

August 1990 page 72
Breakpoint - Basic09 Interfacing.
Greg Law

August 1990 page 38
KISSable OS9 - CoCo 3 does windows.
Dale L. Puckett

September 1990 page 83
If Wife = "Sue," Bring your Potato Salad - Produce
personalized documents with this handy mail-merge
utility.
Christopher Jackson

September 1990 page 28
Breakpoint - Reading and writing.
Greg Law

September 1990 page 39
KISSable OS9 - OS9 gets the grades.
Dale L. Puckett

October 1990 page 44
Displaying Picture Files - Picture storage formats and
displaying images under OS9 Level II.
Tim Kientzle

October 1990 page 30
Breakpoint - Easy access.
Greg Law

October 1990 page 58
KISSable OS9 - OS9 graphics primitives.
Dale L. Puckett

November 1990 page 48

Displaying Picture Files, Part II - Run-length decoding techniques and displaying images with OS9 level II.
Tim Kientzle

November 1990 page 27
KISSable OS9 - Automating the online experience.
Dale L. Puckett

December 1990 page 34
Displaying Pictures Using OS9 Level II Graphics, Part III - Another look at data- compression techniques.
Tim Kientzle

December 1990 page 41
Alarm - A look at sounding the alarm under OS9.
David P Boynton

December 1990 page 87
Breakpoint - 'tis the C zen.
Greg Law

December 1990 page 21
KISSable OS9 - Legends of the C.
Dale L. Puckett

oooooooooooo0000000000oooooooooooo

Converting Amiga 8SVX sound samples for compatibility with PLAY

With the proliferation of MODEMS among CoCo owners, comes the likelihood of acquiring files from other computers. Sometimes these files can be considerable fun to use under OS9 on the CoCo. The files I have been able to make use of are the IFF 8SVX sound sample files from the Commodore Amiga. IFF stands for Interchange File Format, and 8SVX stands for 8 bit Sampled Voice. Most Amiga sound sample files come as IFF format, and have a 'header' of some 60 bytes tacked onto the

beginning of each file to identify it. To allow the PD 'play' command to automatically playback these files, this header must be stripped off, and replaced by two bytes to tell 'play' that the file is from an Amiga, the first byte, and the playback speed, the second byte. To this end I created the following programme conv8svx in C. Compile it using the normal C compiler usage. No special headers or libraries are required.

Bob Devries.

```
/* Convert 8SVX sound samples from Amiga files */
/* Copyright (c) 1992, by Bob Devries */
/* Permission is hereby granted for the non-profit */
/* distribution of this programme as long as the */
/* source code is included, and this header is left */
/* intact */
```

```
#include <stdio.h>
```

```
struct form8svx {
    char form[4];
    long flen;
    char svx[4];
};
struct head {
    char header[4];
    long blklen;
};
struct vce8 {
    long oneshot;
    long repeat;
    long samphi;
    int speed;
    char octave;
    char comp;
    long volume;
};
```

```
main(argc, argv)
```

```

int argc;
char *argv[];
{
    struct form8svx frm8;
    struct head hdr;
    struct vce8 voice8;
    int ch;
    long chunklen;
    char name[256];
    FILE *ifp, *ofp, *fopen();

    setbuf(stdin,NULL); /* set buffer to NULL so single key works */
    setbuf(stdout,NULL);
    pflinit();

    if(argc != 3) { /* silly user tell him how */
        printf("Usage: %s <infile> <outfile>\n",argv[0]);
        exit(1);
    }
    if(access(argv[2],27) != -1) { /* file exists - permission OK? */
        printf("\nFile '%s' exists ! Overwrite Y/N %c",argv[2],7);
        ch = getchar();
        putchar('\n');
        if (toupper(ch) != 'Y') exit(218);
    }
    if((ifp = fopen(argv[1],"r")) == NULL) { /* problem reading file */
        printf("Can't open %s\n",argv[1]);
        exit(errno);
    }
    if((ofp = fopen(argv[2],"w")) == NULL) { /* problem creating file */
        printf("Can't create %s\n",argv[2]);
        exit(errno);
    }
    /* if the file is less than 12 bytes long it's WRONG! */
    if(fread(&frm8,sizeof(frm8),1,ifp) != 1) {
        closeall(ifp,ofp,argv[2]);
        printf("Incorrect file type in %s\n",argv[1]);
        exit(1);
    }
    /* if the word FORM is not present then illegal file type */
    if((strcmp(frm8.form,"FORM") != 0) && (strcmp(frm8.svx,"8SVX") != 0)) {
        closeall(ifp,ofp,argv[2]);
        printf("%s is not an 8SVX sample file!\n",argv[1]);
        exit(1);
    }
    while(!feof(ifp)) {
        if(fread(&hdr,sizeof(hdr),1,ifp) == NULL) { /* get chunk header */
            closeall(ifp,ofp,argv[2]);
            printf("Error reading %s\n",argv[1]);
            exit(errno);
        }
        if((hdr.blklen & 1L) == 1L) hdr.blklen++; /* 68000 pads to even len */
        printf("Chunk %s of length %ld bytes.\n",hdr.header,hdr.blklen);
        if (strcmp(hdr.header,"BODY") == 0) break; /* BODY chunk */
        if (strcmp(hdr.header,"NAME") == 0) {
            if(fread(name,(int)hdr.blklen,1,ifp) == NULL) {
                closeall(ifp,ofp,argv[2]);
            }
        }
    }
}

```



```

        printf("Error reading %s\n",argv[1]);
        exit(errno);
    } else {
        printf("Name = %s\n",name);
        continue;
    }
}
if (strcmp(hdr.header,"VHDR") == 0) { /* Voice8Header */
    if(fread(&voice8,sizeof(voice8),1,ifp) == NULL) {
        closeall(ifp,ofp,argv[2]);
        printf("Error reading %s\n",argv[1]);
        exit(errno);
    } else {
        continue;
    }
}
if(fseek(ifp,hdr.blklen,1) == EOF) {
    closeall(ifp,ofp,argv[2]);
    printf("Error reading %s\n",argv[2]);
    exit(1);
}
}
putc(0x80,ofp);
putc((char){voice8.speed/250},ofp);
chunklen = hdr.blklen;
do {
    ch = getc(ifp);
    putc(ch,ofp);
} while (--chunklen > 0);
fclose(ifp);
fclose(ofp);
printf("Done .... %c\n",7);
}

closeall(in,out,name)
FILE *in;
FILE *out;
char *name;
{
    fclose(in);
    fclose(out);
    unlink(name);
}

```

oooooooooooo0000000000oooooooooooo

VerDisk - Disk verify command for OS9

VerDisk was created from a need to sometimes be able to check whether a disk had errors on it, without going to the rather lengthy trouble of reading all the files on it, one by one.

VerDisk first opens the target disk as a directory, and reads the PD_OPTS of the path to tell whether it is an RBF device.

Next VerDisk reads LSN0 of the target disk to determine how many sectors to read and then reads them one by one, and reports if any errors were found. It will not quit on error until the last sector is read.

VerDisk can take multiple device names on the command line, and will proceed to check each one of them.

Bob Devries

/* VerDisk - Verify integrity of disk sectors */

```

/*      Does a sector read for every sector */
/*      of the target disk.                  */
/* Copyright (c) 1992. By Bob Devries.      */
/* Freely distributable for non-profit only. */

#include <stdio.h>
#define ERROR (-1)

/* The following struct is included here because the one */
/* supplied in 'sgstat.h' does not have an identifier, and */
/* so cannot be used. You may care to modify the relevant */
/* part.                                                    */

struct rbfopt {
    char sg_class,    /* device class - repeated from above */
        sg_drive,    /* drive number */
        sg_step,     /* step rate */
        sg_dtype,    /* device type */
        sg_dense;    /* density capability */
    int  sg_cyls;     /* number of cylinders (tracks) */
    char sg_sides,    /* number of sides */
        sg_verify;   /* 0 = verify on writes */
    int  sg_spt,      /* default sectors per track */
        sg_spt0;     /* ditto track 0 */
    char sg_intlv,    /* sector interleave factor */
        sg_salloc,   /* segment allocation size */
        sg_tfm;      /* dma transfer mode */
    int  sg_exten;    /* path extension for record locking */
    char sg_xxxx,     /* junk fill */
        sg_att,      /* file attributes */
        sg_fdspn[3], /* file descriptor PSN */
        sg_dipn[3];  /* file's directory PSN */
    long sg_dirptr;   /* directory entry pointer */
    int  sg_dvt;     /* address of device table entry */
};

union disksize {
    long numsec;
    char tsec[4];
};

main(argc, argv)
int argc;
char *argv[];
{
    char sector[256];
    char devname[32];
    long pos;
    int i = 1;
    union disksize dsiz;
    struct rbfopt *rbf;
    int op;
    FILE *fp;
    pflinit(); /* we'll be printing longs */
    dsiz.tsec[0] = '\0';

    if(!--argc) { /* dumb user, tell him how it's done */
        printf("Usage: %s /devname [/devname] ...\n", argv[0]);
    }
}

```

```

        exit(215);
    }
    while(argc--) {
        printf("%s: Checking %s\n",argv[0],argv[i]);
        strcpy(devname,argv[i]);
        if((fp = fopen(devname,"d")) == NULL) { /* open as DIR */
            printf("%s: Can't open %s\n",argv[0],argv[i]);
            i++;
            continue; /* get next devname if possible */
        }
        getstat(0,fp,rbf); /* get PD OPTS section of path */
        if(!(rbf->sg class != 1)) { /* wrong device type */
            printf("%s is not an RBF device\n",argv[i]);
            close(fp);
            i++;
            continue;
        }
        fclose(fp);
        strcat(devname,"@");
        op = open(devname, READ); /* had to do open() to do rawread */
        lseek(op,0l,0); /* seek to start of disk and */
        if((read(op,dsiz.tsec+1,3)) == EOF) { /* read first 3 bytes */
            printf("%s: Could not get disk size of %s\n",argv[0],argv[i]);
            close(fp);
            i++;
            continue;
        }
        lseek(op,0l,0); /* seek back to start */
        for(pos=0;pos < dsiz.numsec;pos++) {
            if(read(op,sector,256) == ERROR) { /* read sectors */
                printf("Sector %ld (%$IX) on %s is bad\n",pos,pos,argv[i]);
            }
        }
        close(op); /* finished with this one */
        i++; /* point to next */
    }
}
/* EOF */

```

oooooooooooooooooooooooooooooooo

MORE - From Boisy G. Pitre

Ok guys and gals, here's the newest edition of my 'more' utility. This one handles multiple files and has a few extra options. Hope you like it.

Enjoy!


```

* More - Prompts lists a file or files one
* screen at a time.
* If no files are specified, STDIN is used.
*
* At the --More-- prompt, press:
* <ENTER> to go advance one line
* <BREAK> or 'Q' to quit

```

```

* <SPACE> or any other key to advance
* one screenful
* Usage: More [-l -w] [file] [...]
* -l = show the name of the file before viewing
* (handy for multiple files)
* -w = don't allow lines to wrap around.
* This option truncates the line to a
* length of window's X size - 1.
*
* If you are using a terminal other than the
* OS-9 Level II windowing system, you will need
* to change the reverse on/off sequence as well
* as the clear line sequence

```

AUSTRALIAN OS9 NEWSLETTER

* NOTE: More works great with Shell+'s wildcards!
 * It also works well with external terminals.
 * Just change the Reverse on/off and DelLine
 * bytes to match your terminal's codes.
 * If you are running 'more' on a terminal,
 * it assumes an 80x24 terminal screen size.

* By: Boisy G. Pitre
 * 1204 Love Street
 * Brookhaven, MS 39601
 * Internet: bgpitre@seabass.st.usm.edu

```
ifpl
use /DD/DEFS/os9defs
endc
```

* Terminal specific equates:

```
XSIZE equ 80
YSIZE equ 24
DELNE equ $3
REVON equ $1f20
REVOFF equ $1f21
```

```
mod Size,Name,Prgrm+Objct,Reent+1,Start,Fin
Name fcs /M/
Ed fcb 2

Path rmb 1
Response rmb 1
XH rmb 1
XL rmb 1
YH rmb 1
YL rmb 1
Lflag rmb 1
Buffer rmb 250
FileBuf rmb 60
Stack rmb 200
```

* I make the Parns buffer large in case
 * the wildcard expansion is long,
 * else the system crashes. You can
 * alternately use the shell's memory
 * modifier (i.e. #4k) to insure a big buffer.

```
Parns rmb 4096
```

```
Fin equ .
```

```
Message fdb REVON Reverse Video on
fcc /--More--/
fdb REVOFF Reverse Video off
MessLen equ *-Message
```

```
Header fdb $0a0d
fcc /***** File: /
```

```
HeadLen equ *-Header
```

```
DelLine fcb DELNE Delete line char
```

```
CR fcb $0d Carriage Return

Start pshs x put away X temporarily,
leax IntSvc,pc point to the interrupt
service routine
* os9 F$lcpt and make the system aware of
it

puls x then get X back for processing
clr Path Clear the path (assume stdin)
clr LFlag
lbrs GetSize
* Parsing of the line is done here
Parse lda ,x+
cmpa #$20
beq Parse
cmpa #'-
beq GetOpt
cmpa #$0d
bne OpenIt
tst Path
beq Cycle
lbra Done
```

```
GetOpt lda ,x+
cmpa #$20
beq Parse
anda #$df
cmpa #'L
bne IsItW
com LFlag
bra Parse
IsItW cmpa #'W
lbne Done
lda XL
deca
sta XH
bra Parse
```

* Here, we test to see if the -l
 * flag is set (to display the file
 * header) If so, we print it, else
 * we continue with reading...

```
OpenIt leax -1,x
tst LFlag
beq Open2
pshs x
lbrs PutHead
puls x
dec YH Decrement counter twice to take into
dec YH account the header (two lines)
```

```
Open2 lda #Read. Prepare for reading
os9 I$Open Then open the file
lbcs Error Exit on error
pshs x Save X for later use
sta Path ...else save the path
bra ReadLin and read the line
```

AUSTRALIAN OS9 NEWSLETTER

<pre> Cycle lda YL Get the low order byte sta YH and use the high as a counter bsr PutCR ReadLine lda Path Get the path ldy #250 max chars read = 250 leax Buffer,u point to the buffer os9 I\$ReadLn and read the line bcs EOF if error, check for EOF tst XH Is high order byte set? beq WriteOut Nope, continue as normal pshs x else loop until end of the ldb XH string and place a CR at the Loop leax 1,x end. decb This is unnecessary if the line bne Loop is less than XH, but doesn't slow lda #\$0d down the processing considerably sta ,x and would take longer if we actually puls x checked to see if a CR existed. WriteOut lda #1 Prepare to write to stdout os9 I\$WritLn Write! bcs Error if error, leave dec YH else decrement the counter bne ReadLine if not 0, more lines to show leax Message,pc Prepare to show message ldy #MessLen lda #2 to stderr... os9 I\$Write write it! bcs Error lda #2 Now get response ldy #1 of one character leax Response,u from stderr os9 I\$Read bcs Error bsr KillLine bra TestInp PutCR leax CR,pc lda #1 ldy #1 os9 I\$Write bcs Error rts KillLine lda #2 Here we send a delete line char ldy #1 to clean the prompt. leax DelLine,pc os9 I\$Write bcs Error rts EOF cmpb #E\$EOF Check for end-of-file bne Error tst Path If the path is stdin, we can quit beq Done </pre>	<pre> puls x else there may be more files lbra Parse on the command line. Done clrb Error os9 F\$Exit TestInp lda Response Here we test * the response at prompt cmpa #\$0d is it cr? beq OneLine yep, go up one line anda #\$df else mask uppercase cmpa #'Q is it Q? lbne Cycle nope, * must be space or other char bra Done else we quit IntSvc bsr KillLine Interrupt service routine bra Done OneLine lda #1 We go here if <ENTER> was pressed sta YH,u to increment only one line lbra ReadLine * Here, we actually print the header * for the file we are working on. PutHead pshs x leax Header,pcr ldy #HeadLen lda #1 os9 I\$Write bcs Error puls x bsr SaveFile lda #1 leax FileBuf,u ldy #60 os9 I\$WritLn bcs Error rts ***** * Saves filename in buffer * * Entry: X - Address where filename is * * Exit: None. Filename is stored in FileBuf * SaveFile pshs x leax FileBuf,u SaveF2 lda ,x+ cmpa #\$20 bne SaveF3 lda #\$0d SaveF3 sta ,y+ cmpa #\$0d bne SaveF2 puls x </pre>
--	--

--- **AUSTRALIAN OS9 NEWSLETTER** ---

```

      rts
GetSize pshs  x
      lda    #1      Using stdout...
      ldb    #$26
      os9     I$GetStt Find the size of window
      bcs    ChkErr
      stx     XH      Save the X value
      sty     YH      Save the Y value
      clr     XH      Clear high-order byte of X
      dec     XL      Decrement the X value
      dec     YL      Decrement the Y value
      dec     YL      and dec Y again
      lda     YL      Do the initial load
      sta     YH      of the counter
      puls    x

```

```

      rts
      * If this is true, then we are probably dealing with a
      * terminal, not a hardware window.
      * We'll assume 80x24 as the terminal size.
ChkErr cmpb    #E$UnkSvc
      bne     Error
      lda     #XSIZE
      sta     XL
      lda     #YSIZE
      sta     YL
      clr     XH
      rts
      emod
Size    equ    *
      end

```

oooooooooooo0000000000oooooooooooo

Scribe -- Selected messages from the Os9 area on 05-05-1992

Good news to all you HD owner who can't seem to find the time HDKIT requires to backup your hard drive.....
Bring on "STREAM".....

***** ABOUT FOUR TIMES FASTER THAN HDKIT *****

Some time ago, Bruce Isted wrote a backup program called STREAM. It has since been tested extensively by five or six beta testers. I'm pleased to tell you that it is now available as of about a week ago when Kim Bergman phoned Bruce and asked if it was OK to add it to the OCN Library. It is also available on the Keyboard BBS (see tag line) as of this day.

HD Kit was and still is a great program but some what SLOW. Bruce still recomends using Pete Lyall's "Files" module from HD Kit or D.P. Johnson's "LS" command, with STREAM.

Here is a comparison between the two programs using a 20 meg HD and THREE types of disk drives (1440 sect., 2880 sect & 1.44 meg).

STATS ON STREAM VS HD KIT (using a 20 meg ST-225 Hard Disk Drive)

```

=====
Std disk Drives !! 80 tk disk drives !! 1.44 meg disk dr.
1440 Sector Drives!! 2880 Sector Drives!! 5760 Sector Drives
.....!!.....!!.....!!
disks Backup TIME!!disks Backup TIME!!disks Backup TIME:
used 1 disk TOTAL!!used 1 disk TOTAL!!used 1 disk TOTAL
.....!!.....!!.....!!
55 9.5min 523min!! 28 19.min 535min!! not tried via HD KIT
.....!!.....!!.....!!
55 1.8min 99min!! 28 4.min 112min!! 14 8.min 112min via STREAM
.....!!.....!!.....!!

```

oooooooooooo0000000000oooooooooooo